INFO-F-307 – Groupe 5

Première Itération – vue d'ensemble de l'architecture de l'application

1. Patron Singleton :

Nous avons décidé que pour l'histoire « liste de shopping », l'utilisateur ne pourrait avoir qu'une seule liste de shopping sur son téléphone. Nous avons donc utilisé le patron singleton pour réaliser cette liste. Avoir une liste unique utilisant le patron singleton a eu pour avantages :

- Eviter de devoir passer en argument la liste courante à chaque endroit où elle est utilisée.
- Eviter de devoir demander à chaque ajout de produit, dans quelle liste l'utilisateur veut rajouter son produit.
- L'utilisation du patron Singleton à la place d'utiliser une liste statique par exemple, à permis de pouvoir effectuer des tests plus aisément en créant notamment une « fausse » instance (voir shoppingList).

2. Patron MVVM :

2.1. Description MVVM

Pour notre architecture globale, nous avons utilisé le patron MVVM :

- **M** pour **M**odel (**M**odèle) : représente les objets venant des données.
- **V** pour **V**iew (**V**ue) : représente les interfaces graphiques (Activity + xml)
- **VM** pour **V**iew-**M**odel (**M**odèle de la **V**ue) : représentes les données des l'interfaces graphiques, ce qui est affiché ainsi que les interactions avec celles-ci.

Pour un objet description de notre modèle par exemple (représente tout objet ayant un nom, une description, un ID et éventuellement par après une image) :

- **M** : models.Description
- **V** : app.views.DescriptionActivity (+layout)
- **VM** : app.views.model.DescriptionActivityModel

Le modèle d'une vue est bien entendu la composée de modèles de données, dont il s'occupe parfois de veiller à la conversion.

Pour implémenter une telle architecture, nous avons utilisé une libraire : Android Binding, permettant de réaliser ce patron aisément.

2.2. Android Binding

L'Android Binding s'utilise en rajoutant directement dans les layouts xml des attributs « binding ».
Exemple :

```
<LinearLayout
 xmlns:android=http://schemas.android.com/apk/res/android
 [...]
 binding:onlclick="doSomething" >
```

Ils permettent notamment de faire office de contrôleur grâce à des événements comme onclick, onTextChanged,... Ceux-ci permettent d'appeler directement des commandes. Ces commandes sont à implémenter dans le modèle de la vue. Cela permet d'éviter d'écrire tout les code d'interaction pour une activité et de ce concentrer sur l'action en elle même.

Ces attributs permettent également de lier la valeur courante d'une zone de texte par exemple à un observable qui, s'il est modifié dans le modèle de la vue sera automatiquement mis à jour dans la vue.
Exemple :

```
public Observable<String> Name = new StringObservable(«...»);
```

Ces observables peuvent aussi être de type ArrayList et être utilisé pour une « listView » par exemple ou encore de type booléen pour afficher ou cacher une partie de la vue de façon dynamique.

La liaison modèle de la vue – vue se fait dans les activités grâce à une simple procédure :

Object ViewModel;
ViewModel = new ModelName(/*paramètres*/); -> on crée le modèle
setAndBindRootView(R.layout.layoutName,ViewModel) -> on lie le modèle à la vue

3. Interface avec la base de données (packages store.*)

Pour se simplifier la tâche, l'implémentation de l'interface avec la base de données à été écrite de façon à directement remplir un ObservableArray<T>. Cela a permit ainsi de les utiliser directement dans les modèles de la vue. Pour ce faire un héritage de ArrayListObservable<T> a été implémenté dans DBlist.java.

4. Packages

On a trois grand groupes de packages :

app.* : packages pour l'application et ces activités ainsi que les modèles de la vue.
models.* : packages incluant notre modèle de données et leurs interfaces
store.* : packages permettant la liaison avec la base de données.